Turbo SHA-2

Danilo Gligoroski and Svein Johan Knapskog

Centre for Quantifiable Quality of Service in Communication Systems, Norwegian University of Science and
Technology, O.S.Bragstads plass 2E, N-7491 Trondheim, NORWAY
{Danilo.Gligoroski,Svein.J.Knapskog}@q2s.ntnu.no

Abstract. In this paper we describe the construction of Turbo SHA-2 family of cryptographic hash functions. They are built with design components from the SHA-2 family, but the new hash function has three times more chaining variables, it is more robust and resistant against generic multi-block collision attacks, its design is resistant against generic length extension attacks and it is 2 - 8 times faster than the original SHA-2. It uses two novel design principles in the design of hash functions: *1. Computations in the iterative part of the compression function start by using variables produced in the message expansion part that have the complexity level of a random Boolean function, 2. Variables produced in the message expansion part are not discarded after the processing of the current message block, but are used for the construction of the three times wider chain for the next message block.* These two novel principles combined with the already robust design principles present in SHA-2 (such as the nonlinear message expansion part), enabled us to build the compression function of Turbo SHA-2 that has just 16 new variables in the message expansion part (compared to 48 for SHA-256 and 64 for SHA-512) and just 8 rounds in the iterative part (compared to 64 for SHA-256 and 80 for SHA-512).

Key words: Cryptographic hash function, SHA-2, Turbo SHA-2

1 Introduction

Recently we have witnessed several significant breakthroughs in the general understanding of cryptographic hash functions, such as those of Joux [14] in 2004 where he constructed a new generic multi–collisions attack, and those of Kelsy and Schneier [15] in 2005 where they extended Joux' approach for finding expandable messages with different length. In 2005 Coron at al. [5] made several suggestions for how to strengthen the Merkle-Damgård design [19, 6], and at the same time Gauravaram, Millan and Neito [10] gave an interesting discussion about the possibilities that Merkle-Damgård design for the the concrete family of cryptographic hash functions (the family of MD4) was in fact not properly implemented. That is especially true for the so called pseudo-collisions attack, for which MD5, SHA-0 and SHA-1 were not designed to be secure. These thoughts for the design criteria are also present in Preneel's works [25, 27].

Moreover, for a period of 15 years we have witnessed several breakthroughs in cryptanalysis and successful attacks on concrete cryptographic hash functions of the MD4 family. We will mention some of them: den Boer and Bosselaers [2, 3] in 1991 and 1993, Vaudenay [29] in 1995, Dobbertin [7] in 1996 and 1998, Chabaud and Joux [4] in 1998, Biham and Chen [1] in 2004, and Wang et al. [30–33] in 2005. Most well known cryptographic hash functions such as: MD4, MD5, HAVAL, RIPEMD, SHA-0 and SHA-1, have succumbed to those attacks.

The SHA-2 family of hash functions was designed by NSA and adopted by NIST in 2000 as a standard that is intended to replace SHA-1 in 2010 [20]. Several papers have been devoted for cryptanalysis of SHA-2 hash functions. Here we will mention some of them. Gilbert and Handschuh in 2003 have made an analysis of the SHA-2 family [11]. They proved that there exist XOR-differentials that give a 9-round local collision with probability 2^{-66}. In 2004, Hawkes, Paddon and Rose [13] improved the result and showed existence of addition-differentials of 9-round local collisions with probability of 2^{-39}. In 2005, Yoshida and Biryukov analyzed a variant of SHA-256

[34] where they replaced every arithmetic addition by XOR operation. In 2006, Mendel et al. [17], found XOR-differentials for 9-round local collisions, also with probability 2^{-39}.

Following the developments in the field of cryptographic hash functions, NIST organized two cryptographic hash workshops [21] in 2005 and 2006 respectively. As a result of those workshops, NIST decided to run a 4 year hash competition for selection of a new cryptographic hash standard [22]. The requirements for the hash digest size for the new cryptographic hash functions are: 224, 256, 384 and 512 bits - the same as for the current SHA-2 standard.

Our Work: By introducing two novel design principles in the design of hash functions, and by using components from the SHA-2 family, we describe the design of a new family of cryptographic hash functions called Turbo SHA-2. These two novel principles are:

1. *Computations in the iterative part of the compression function start by using variables produced in the message expansion part that have the complexity level of a random Boolean function.*
2. *Variables produced in the message expansion part are not discarded after the processing of the current message block, but are used for the construction of the three times wider chain for the next message block.*

These two novel principles combined with the already robust design principles present in SHA-2 (such as nonlinear dependency in the message expansion part), enabled us to build a compression function of Turbo SHA-2 that has the following properties:

1. The message expansion part has just 16 new variables (compared to 48 for SHA-256 and 64 for SHA-512).
2. The iterative part has just 8 rounds (compared to 64 for SHA-256 and 80 for SHA-512).
3. The function has three times more chaining variables.
4. The new hash function is more robust and resistant against generic multi-block collision attacks.
5. The new hash function is resistant against generic length extension attacks.
6. The new hash function is 2 - 8 times faster than original SHA-2.

The paper is organized as follows. Some preliminaries are given in Section 2. The algorithm Turbo SHA-2 is given in Section 3, security analysis is given in Section 4, in Section 5 we give a speed comparison of optimized C implementations of SHA-2 and Turbo SHA-2 and in Section 6 we close our paper with conclusions.

2 Preliminaries and notation

In this paper we will use the same notation as that of NIST: FIPS 180-2 description of SHA-2 [20].

The following operations are applied to 32-bit or 64-bit words in Turbo SHA-2:

1. Bitwise logical word operations: '$\wedge$' – AND , '$\vee$' – OR, '$\oplus$' – XOR and '$\neg$' – Negation.
2. Addition '+' modulo 2^{32} or modulo 2^{64}.
3. The shift right operation, $SHR^n(x)$, where x is a 32-bit or 64-bit word and n is an integer with $0 \le n < 32$ (resp. $0 \le n < 64$).
4. The rotate right (circular right shift) operation, $ROTR^n(x)$, where x is a 32-bit or 64-bit word and n is an integer with $0 \le n < 32$ (resp. $0 \le n < 64$).

Depending on the context we will sometimes refer to the hash function as Turbo SHA-2, and sometimes as Turbo SHA-224/256 or Turbo SHA-384/512.

Turbo SHA-224/256	Turbo SHA-384/512
$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z),$	$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z),$
$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z),$	$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z),$
$\sum_0^{\{256\}} = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x),$	$\sum_0^{\{512\}} = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x),$
$\sum_1^{\{256\}} = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x),$	$\sum_1^{\{512\}} = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x),$
$\sigma_0^{\{256\}} = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x),$	$\sigma_0^{\{512\}} = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x),$
$\sigma_1^{\{256\}} = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x).$	$\sigma_1^{\{512\}} = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x).$

Table 1. Logical functions for Turbo SHA-2

2.1 Turbo SHA-2 logical functions

Turbo SHA-2 uses the same six logical functions as SHA-2 (listed in Table 1). Each function operates on 32-bit (resp. 64-bit) words, which are represented as x, y, and z. The result of each function is a new 32-bit (resp. 64-bit) word.

2.2 Turbo SHA-2 Constants

Turbo SHA-2 does not use any constants.

2.3 Preprocessing

Preprocessing in Turbo SHA-2 is exactly the same as that of SHA-2. That means that these three steps: padding the message M, parsing the padded message into message blocks, and setting the initial hash value, $H^{(0)}$ are the same as in SHA-2. Thus in the parsing step the message is parsed into N blocks of 512 bits (resp. 1024 bits), and the i-th block of 512 bits (resp. 1024 bits) is a concatenation of sixteen 32-bit (resp. 64-bit) words denoted as $M_0^{(i)}, M_1^{(i)}, \ldots, M_{15}^{(i)}$.

Turbo SHA-224/256 may be used to hash a message, M, having a length of l bits, where $0 \leq l < 2^{64}$, while Turbo SHA-384/512 may be used to hash a message, M, having a length of l bits, where $0 \leq l < 2^{128}$.

2.4 Initial Hash Value $H^{(0)}$

The initial hash value, $H^{(0)}$ for Turbo SHA-2 is the same as that of SHA-2 (given in Table 2).

Turbo SHA-224	Turbo SHA-256	Turbo SHA-384	Turbo SHA-512
$H_0^{(0)} =$ c1059ed8,	$H_0^{(0)} =$ 6a09e667,	$H_0^{(0)} =$ cbbb9d5dc1059ed8,	$H_0^{(0)} =$ 6a09e667f3bcc908,
$H_1^{(0)} =$ 367cd507,	$H_1^{(0)} =$ bb67ae85,	$H_1^{(0)} =$ 629a292a367cd507,	$H_1^{(0)} =$ bb67ae8584caa73b,
$H_2^{(0)} =$ 3070dd17,	$H_2^{(0)} =$ 3c6ef372,	$H_2^{(0)} =$ 9159015a3070dd17,	$H_2^{(0)} =$ 3c6ef372fe94f82b,
$H_3^{(0)} =$ f70e5939,	$H_3^{(0)} =$ a54ff53a,	$H_3^{(0)} =$ 152fecd8f70e5939,	$H_3^{(0)} =$ a54ff53a5f1d36f1,
$H_4^{(0)} =$ ffc00b31,	$H_4^{(0)} =$ 510e527f,	$H_4^{(0)} =$ 67332667ffc00b31,	$H_4^{(0)} =$ 510e527fade682d1,
$H_5^{(0)} =$ 68581511,	$H_5^{(0)} =$ 9b05688c,	$H_5^{(0)} =$ 8eb44a8768581511,	$H_5^{(0)} =$ 9b05688c2b3e6c1f,
$H_6^{(0)} =$ 64f98fa7,	$H_6^{(0)} =$ 1f83d9ab,	$H_6^{(0)} =$ db0c2e0d64f98fa7,	$H_6^{(0)} =$ 1f83d9abfb41bd6b,
$H_7^{(0)} =$ befa4fa4.	$H_7^{(0)} =$ 5be0cd19.	$H_7^{(0)} =$ 47b5481dbefa4fa4.	$H_7^{(0)} =$ 5be0cd19137e2179.

Table 2. The initial hash value, $H^{(0)}$ for Turbo SHA-2

2.5 Initial double pipe $P^{(0)}$

Turbo SHA-2 has an additional chaining pipe $P^{(i)}$ that has length which is two times longer than the hash digest chain. That means that there are sixteen 32-bit (resp. 64-bit) chaining variables in $P^{(i)}$. Their initial values are the same as the first sixteen constants in the SHA-2 design. They are given below (from left to right):

For Turbo SHA-224/256

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
```

and for Turbo SHA-384/512

```
428a2f98d728ae22 7137449123ef65cd b5c0fbcfec4d3b2f e9b5dba58189dbbc
3956c25bf348b538 59f111f1b605d019 923f82a4af194f9b ab1c5ed5da6d8118
d807aa98a3030242 12835b0145706fbe 243185be4ee4b28c 550c7dc3d5ffb4e2
72be5d74f27b896f 80deb1fe3b1696b1 9bdc06a725c71235 c19bf174cf692694
```

2.6 Turbo SHA-2 Hash Computation

The Turbo SHA-2 hash computation uses functions and initial values defined in previous subsections. So, after the preprocessing is completed, each message block, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$, is processed in order, using the steps described algorithmically in Table 3.

The algorithm uses 1) a message schedule of thirty-two 32-bit (resp. 64-bit) words, 2) eight working variables of 32 bits (resp. 64 bits), 3) a hash value of eight 32-bit (resp. 64-bit) words and 4) additional double pipe chain of sixteen 32-bit (resp. 64-bit) words. The final result of Turbo SHA-256 is a 256-bit message digest and of Turbo SHA-512 is a 512-bit message digest. The final result of Turbo SHA-224 and Turbo SHA-384 are also 256 and 512 bits, but the output is then truncated as in SHA-2 to 224 (resp. 384 bits). The words of the message schedule are labeled $W_0, W_1, \ldots, W_{31}$. The eight working variables are labeled a, b, c, d, e, f, g, and h and sometimes they are called “state register”. The words of the hash value are labeled $H_0^{(i)}$, $H_1^{(i)}$, …, $H_7^{(i)}$, which will hold the initial hash value, $H^{(0)}$, replaced by each successive intermediate hash value (after each message block is processed), $H^{(i)}$, and ending with the final hash value, $H^{(N)}$. The words of the additional double pipe chain are labeled $P_0^{(i)}$, $P_1^{(i)}$, …, $P_{15}^{(i)}$, which will hold the initial double pipe value, $P^{(0)}$, replaced by each successive intermediate double pipe value (after each message expansion for each message block), $P^{(i)}$. Turbo SHA-2 also uses two temporary words, T_1 and T_2. Note that in the description of the algorithm, instead of using the notation $\sigma_0^{\{256\}}$, $\sigma_1^{\{256\}}$, $\sigma_0^{\{512\}}$ or $\sigma_1^{\{512\}}$ which depends on whether we are working with 32-bit or 64-bit words, we simply use σ_0 and σ_1.

3 Security of Turbo SHA-2

In this section we will make an initial analysis of how strongly collision resistant, preimage resistant and second preimage resistant Turbo SHA-2 is. We will start by describing our design rationale, then we will analyze the properties of the message expansion part and finally we will discuss the strength of the function against known attacks for finding different types of collisions.

```
For $i = 1$ to $N$:
{
  1. Message expansion part for obtaining additional sixteen 32-bit (64-bit) words:

$W_t = \begin{cases} M_t^{(i)}, & 0 \le t \le 15 \\ W_{t-16} + \sigma_0(W_{t-15}) + W_{t-14} + \sigma_1(W_{t-13}) + W_{t-12} + \sigma_0(W_{t-11}) + W_{t-10} + \sigma_1(W_{t-9}) + \\ +W_{t-8} + W_{t-7} + \sigma_0(W_{t-6}) + W_{t-5} + \sigma_1(W_{t-4}) + W_{t-3} + \sigma_1(W_{t-2}) + \sigma_0(W_{t-1}) + \\ +P_{t-16}^{(i-1)}, & 16 \le t \le 31 \end{cases}$

  2. Set the $i^{\text{th}}$ intermediate double pipe value $P^{(i)}$: $P_t^{(i)} = W_t + W_{t+16}, \qquad 0 \le t \le 15$

  3. Initialize eight working variables $a$, $b$, $c$, $d$, $e$, $f$, $g$ and $h$ with the $(i-1)^{\text{th}}$ hash value and the
values of $W_{31}$, $W_{30}$, $W_{29}$, $W_{28}$, $W_{27}$, $W_{26}$, $W_{25}$, $W_{24}$:

      $a = H_0^{(i-1)} + W_{31}, \quad b = H_1^{(i-1)} + W_{30}, \quad c = H_2^{(i-1)} + W_{29}, \quad d = H_3^{(i-1)} + W_{28},$
      $e = H_4^{(i-1)} + W_{27}, \quad f = H_5^{(i-1)} + W_{26}, \quad g = H_6^{(i-1)} + W_{25}, \quad h = H_7^{(i-1)} + W_{24}$

  4. For t=0 to 7
    {
      $T_1 = h + \sum_1(e) + Ch(e, f, g) + (W_t \oplus W_{t+16}) + (W_{t+4} \oplus W_{t+24}) + (W_{t+8} \oplus W_{t+20}) + W_{t+12}$
      $T_2 = \sum_0(a) + Maj(a, b, c)$
      $h = g$
      $g = f$
      $f = e$
      $e = d + T_1$
      $d = c$
      $c = b$
      $b = a$
      $a = T_1 + T_2$
    }

  5. Compute the $i^{\text{th}}$ intermediate hash value $H^{(i)}$:

      $H_0^{(i)} = a + H_0^{(i-1)}, \quad H_1^{(i)} = b + H_1^{(i-1)}, \quad H_2^{(i)} = c + H_2^{(i-1)}, \quad H_3^{(i)} = d + H_3^{(i-1)},$
      $H_4^{(i)} = e + H_4^{(i-1)}, \quad H_5^{(i)} = f + H_5^{(i-1)}, \quad H_6^{(i)} = g + H_6^{(i-1)}, \quad H_7^{(i)} = h + H_7^{(i-1)}$
}
```

Table 3. Algorithmic description of Turbo SHA-2 hash function.

3.1 Design rationale

The reasons for first principle: *Computations in the iterative part of the compression function start by using variables produced in the message expansion part that have the complexity level of a random Boolean function.*

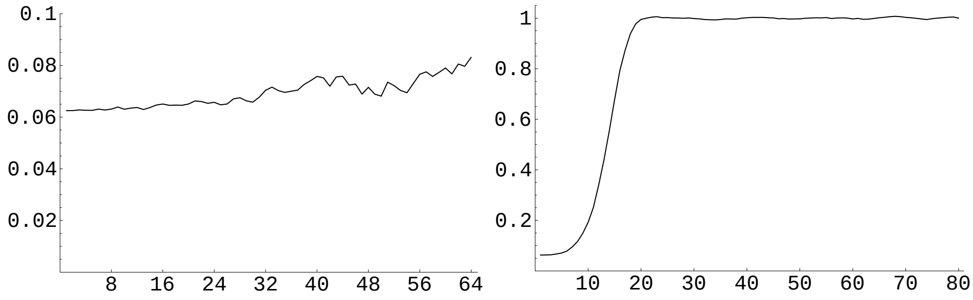
The monomial tests have been introduced several years ago by Foliol [9] to evaluate the statistical properties of symmetric ciphers. Later, Saarinen [28] proposed an extension of Foliol's ideas to a chosen IV statistical attack, called the "d-monomial test", and used it to find weaknesses in several proposed stream ciphers. In 2007 Englund, Johansson and Turan [8] generalized Saarinen's idea and proposed a framework for chosen IV statistical attacks using a polynomial description. Their basic idea is to select a subset of IV bits as variables, assuming all other IV values as well as the key being fixed. Then by obtaining the algebraic normal form for such a function they were searching for some statistical deviations from ideal random Boolean function. A similar approach as that of Englund et al. is also described by O'neil in [23].

In order to get a measure how far or close the parts of a cryptographic hash function are from a random Boolean function we have defined NANT - A Normalized Average Number of Terms (monomials). NANT can be seen as a variant of Englund's monomial tests and it is defined in Appendix 1.

We have measured NANT for SHA-1 and SHA-256 and in Figures 1 and 2 we give graphs for SHA-1 and SHA-256 both for their message expansion part and for their iterative part.

In Figure 1a. it can be seen that the message expansion part of SHA-1, being completely linear, never reaches the complexity of a random Boolean function. In Figure 1b. we can see that SHA-1 reaches the complexity of a random Boolean function after 20 rounds in its iterative part. There are numerous arguments in all successful attacks on SHA-0 and SHA-1 that actually the linearity of their message expansion part is the essential source of their weaknesses.

The situation with SHA-2 is significantly different. From Figure 2a. we see that the message expansion part of SHA-2 is much better designed and it reaches the same complexity as a random Boolean function after 16 rounds, which reflects afterwards in the iterative part of SHA-2 that achieves the complexity level of a random Boolean function after 13 rounds (Figure 2b.) We point out that there is a strong correlation between this relatively slow run of 13 rounds until reaching the level of random Boolean function and the discovery of 9-round local collisions with probability 2^{-66} made by Gilberth and Hanchuch [11], and afterwards improved to the probability 2^{-39} by Hawkes et al. [13] and by Mendel et al. [17].



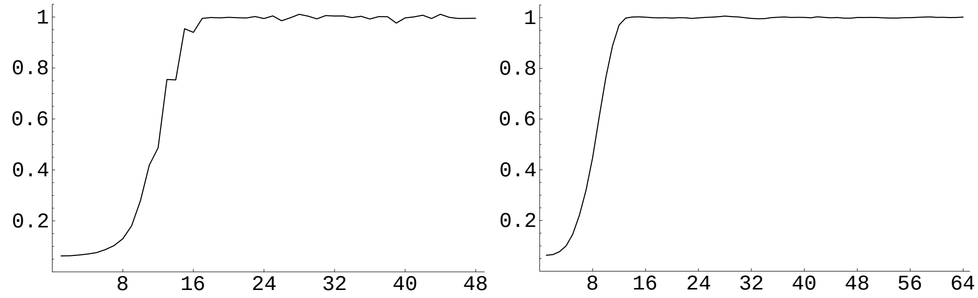
1a. SHA-1 Message Expansion

1b. SHA-1 Iterations

Fig. 1. Monomial test NANT for SHA-1 in the message expansion part and in the iterative part of its compression function.

In Figures 3a. and 3b. we show the complexity levels of our proposal Turbo SHA-2. By introducing our first principle, we have achieved that Turbo SHA-2 starts immediately with a complexity of a random Boolean function in its iterative part. Moreover, the number of rounds in the message expansion part, necessary to reach the level of random Boolean function in Turbo SHA-2 is 4. That means that only four working variables $W_{16}, \ldots, W_{19}$ expressed as Boolean functions are far from acting as a random Boolean function. The remaining twelve working variables $W_{20}, \ldots, W_{31}$ have a complexity level as a random Boolean function.

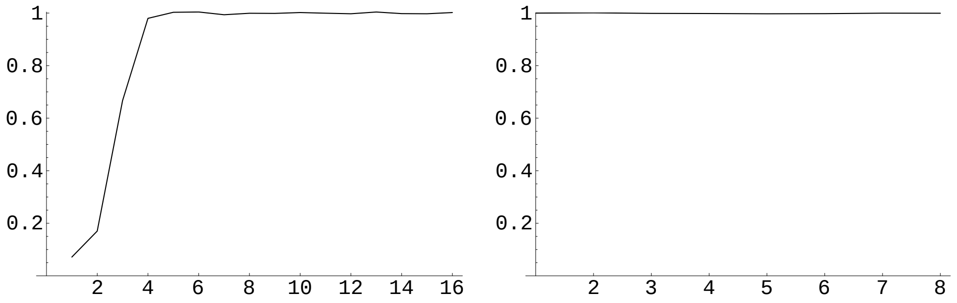
The reasons for the second principle: *Variables produced in the message expansion part are not discarded after the processing of the current message block, but are used for the construction of the three times wider chain for the next message block.*



2a. SHA-2 Message Expansion

2b. SHA-2 Iterations

Fig. 2. Monomial test NANT for SHA-2 in the message expansion part and in the iterative part of its compression function.



3a. Turbo SHA-2 Message Expansion

3b. Turbo SHA-2 Iterations

Fig. 3. Monomial test NANT for Turbo SHA-2 in the message expansion part and in the iterative part of its compression function.

In the design of Turbo SHA-2 we have decided to incorporate the suggestions of Lucks [16] and Coron et al. [5]. Namely, by setting the size of the chaining pipe (equivalent to the internal memory of the iterated compression function) to be three times bigger than the output length, weaknesses against generic attacks of Joux [14], and Kelsy and Schneier [15] are eliminated. One part of that chaining pipe is the classical hash chaining value. Additionally we use the information of the working variables produced in the message expansion part to construct the other two parts of the pipe.

So far, all other designs are using the variables produced in the message expansion part just for the processing of the current message block. Then, all that information is waisted and not used for the next message block. From security reasons we can say that that is a good strategy having in mind that message expansion part in the designs before SHA-2 are producing variables that seen as Boolean functions are not at all close to random Boolean functions. However, SHA-2 and Turbo SHA-2 produce working variables that are similar as random Boolean functions. So, we decided not to waist the expensive computations performed in the message expansion part just for the processing of the current message block, but to use the variables produced in that part for construction of the additional double pipe P.

Having this design principle, Turbo SHA-2 naturally guarantees a resistance against generic length extension attacks, from which almost all members of MD4 family (including also SHA-256 and SHA-512, but not SHA-224 and SHA-384 - due to the final truncation) suffer.

Why Turbo SHA-2 does not have constants? The reasons why we decided not to use any constants in the iterative part of the compression function of Turbo SHA-2 are due to the fact that in the message expansion part, actually we are using sixteen "initial constants" $P^{(0)}$ as the initial values of the double pipe.

Controlling the differentials is much harder in Turbo SHA-2 than in SHA-2! The XOR-differentials that Gilbert and Handschuh found for SHA-256 [11] give a 9-round local collision with probability 2^{-66}. Those differentials are low-weight Hamming differentials. Additionally, for one-bit differentials they set with a probability $\frac{1}{2}$ that the output of the functions Maj and Ch is 0, and also with a probability $\frac{1}{2}$ that the operation addition modulo 2^{32} behaves as XOR. Having these preconditions, from the equations that define the state register update they set the differentials that they considered in their paper as optimal from the perspective of search for low-weight Hamming differentials. Those differentials are the following: If W_i is the word containing the perturbative one-bit difference, then the next eight word differences are $W_{i+1} = \sum_1(W_i) \oplus \sum_0(W_i)$, $W_{i+2} = \sum_1(\sum_0(W_i))$, $W_{i+3} = 0$, $W_{i+4} = W_i$, $W_{i+5} = \sum_1(W_i) \oplus \sum_0(W_i)$, $W_{i+6} = 0$, $W_{i+7} = 0$, $W_{i+8} = W_i$.

Later on, Hawkes et al. [13] agreed that their differentials are optimal, but by having a knowledge of the values of the intermediate values of the state register, and by introducing addition-differentials they improved the result and showed existence of addition-differentials of 9-round local collisions with probability of 2^{-39}. Very similar strategy as that of Hawkes et al., but for finding XOR-differentials have been done by Mendel et al. [17], for 9-round local collisions, also with probability 2^{-39}.

The strategy of Gilbert and Handschuh for definition of optimal differentials can not be directly applied for Turbo SHA-2. That is because of the following reasons:

Iterative part of Turbo SHA-2 starts with the initial assignment of the state register that includes both previous chaining value of the hash, and the values of the last eight working variables: $a = H_0^{(i-1)} + W_{31}$, $b = H_1^{(i-1)} + W_{30}$, $\ldots$, $h = H_7^{(i-1)} + W_{24}$. So, setting up a one-bit difference in the state register actually requires assumption that the message expansion part produced one-bit difference only in one W_i, $i = 24, \ldots, 31$. But then, just in the first four rounds all 32 working variables W_i, $i = 0, \ldots, 31$ are participating in the update of the state register, and the differences between the working variables are far from being a one-bit differences.

3.2 Properties of message expansion part

It is relatively easy to prove the following Theorem:

Theorem 1. *The message expansion part of Turbo SHA-224/256 is a bijection $\xi : \{0,1\}^{512} \rightarrow \{0,1\}^{512}$ and of Turbo SHA-384/512 is a bijection $\xi : \{0,1\}^{1024} \rightarrow \{0,1\}^{1024}$.*

Proof. It is enough to show that the message expansion part is surjection, i.e. for every 16-tuple $W = (W_{16}, W_{17}, \ldots, W_{31})$ there exist a 16-tuple preimage $M = (M_0, M_1, \ldots, M_{15})$ such that $\xi(M) = W$.

First we should note that operations $\sigma_0^{\{256\}}, \sigma_1^{\{256\}} : \{0,1\}^{32} \rightarrow \{0,1\}^{32}$ and $\sigma_0^{\{512\}}, \sigma_1^{\{512\}} : \{0,1\}^{64} \rightarrow \{0,1\}^{64}$ are bijections.

Then, by rearranging the recurrent equation that describes the message expansion part for a given 16-tuple $W = (W_{16}, W_{17}, \ldots, W_{31})$ we have the relation: $W_{31} = W_{15} + \sigma_0(W_{16}) + W_{17} + \sigma_1(W_{18}) + W_{19} + \sigma_0(W_{20}) + W_{21} + \sigma_1(W_{22}) + W_{23} + W_{24} + \sigma_0(W_{25}) + W_{26} + \sigma_1(W_{27}) + W_{28} + \sigma_1(W_{29}) + \sigma_0(W_{30}) + P_{15}^{(i-1)}$. From there it is straightforward to compute the unique value for W_{15}. Now, having the new 16-tuple $W = (W_{15}, W_{16}, \ldots, W_{30})$ we can proceed further to compute the unique value for W_{14}, and so on until we compute the unique value for W_0. □

3.3 Properties of the iterative part

As we have already mentioned in the design rationale, the second principle, guarantees that the variables in the iterative part start immediately with algebraic complexity of a random Boolean function. That fact, further enabled us both to reduce the number of iteration rounds in the hash function, and to thwart any of the current successful attacks on MDx family of hash functions.

In order to achieve mixing of all 32 variables W_i just in the first four rounds Turbo SHA-2 uses the assignment $T_1 = h + \sum_1(e) + Ch(e,f,g) + (W_t \oplus W_{t+16}) + (W_{t+4} \oplus W_{t+24}) + (W_{t+8} \oplus W_{t+20}) + W_{t+12}$. In Table 4 we give the order of how the iterative part of the Turbo SHA-2 combines the variables $W_i,\ 0 \le i \le 31$.

Step	Used W_i							
Initialization	W_{31}	W_{30}	W_{29}	W_{28}	W_{27}	W_{26}	W_{25}	W_{24}
t=0	W_0	W_4	W_8	W_{12}	W_{16}	W_{20}	W_{24}	
t=1	W_1	W_5	W_9	W_{13}	W_{17}	W_{21}	W_{25}	
t=2	W_2	W_6	W_{10}	W_{14}	W_{18}	W_{22}	W_{26}	
t=3	W_3	W_7	W_{11}	W_{15}	W_{19}	W_{23}	W_{27}	
t=4	W_4	W_8	W_{12}	W_{16}	W_{20}	W_{24}	W_{28}	
t=5	W_5	W_9	W_{13}	W_{17}	W_{21}	W_{25}	W_{29}	
t=6	W_6	W_{10}	W_{14}	W_{18}	W_{22}	W_{26}	W_{30}	
t=7	W_7	W_{11}	W_{15}	W_{19}	W_{23}	W_{27}	W_{31}	

Table 4. The order of using the working variables $W_i,\ 0 \le i \le 31$ in the compression function of Turbo SHA-2.

3.4 Finding collisions in variants of the reduced compression function of Turbo SHA-2

In this subsection we will analyze a reduced compression function of Turbo SHA-2 with only one or two rounds, and we will show how to find collisions with workload less than $2^{\frac{n}{2}}$, where $n = 256$ or $n = 512$ only for the reduced function with one round.

Turbo SHA-2 with one round: Instead of the notation $W_i,\ 0 \le i \le 31$, for the variables produced in the message expansion part, we will use the notation: $M_i = W_i$ for $0 \le i \le 15$ and W_i for $16 \le i \le 31$.

With only one round, Turbo SHA-2 does not use all 32 variables defined in the message expansion part. Actually, together with the initialization in which it uses the values $W_{31}, W_{30}, \ldots, W_{24}$ the set of all 32 variables $W_i,\ 0 \le i \le 31$, can be divided into four disjunctive subsets:

1. **Used words from the extension part.** $X_1 = \{W_{16}, W_{19}, W_{24}, \ldots, W_{31}\}$
2. **Used words from the message.** $Y_1 = \{M_0, M_4, M_8, M_{12}\}$
3. **Unused words from the extension part.** $X_2 = \{W_{17}, W_{18}, W_{20}, W_{21}, W_{22}, W_{23}\}$
4. **Unused words from the message.** $Y_2 = \{M_1,\ M_2,\ M_3,\ M_5,\ M_6,\ M_7,\ M_9,\ M_{10},\ M_{11},\ M_{13},\ M_{14},\ M_{15}\}$

Now the search for collisions will go like this:

```
1. Fix the values of the set $X_1$.
2. Repeat
3.     Chose randomly values of the set $X_2$.
```

```
4.     As in Theorem 1 find the solution $M = \{M_0, \ldots, M_{15}\}$.
5. until Found two different sets $X_2'$ and $X_2''$ such that $M_0' = M_0''$ AND $M_4' = M_4''$ AND $M_8' = M_8''$ AND $M_{12}' = M_{12}''$.
```

Note that the stop criteria in this collision search algorithm is the collision over the set $Y_1 = \{M_0, M_4, M_8, M_{12}\}$. Since the total number of bits of the variables in the set Y_1 is $4 \times 32 = 128$ (resp. $4 \times 64 = 256$) from the birthday paradox we can expect that after $\approx 2^{64}$ (resp. $\approx 2^{128}$) attempts of different values from the set X_2 we can find a collision.

For a two round reduced Turbo SHA-2 the above strategy will not give faster collision search than the brute force search. We show that by the following argument:

Turbo SHA-2 with two rounds: For a two rounds reduction we will have the following situation:

1. Used words from the extension part. $X_1 = \{W_{16}, W_{17}, W_{20}, W_{21}, W_{24}, \ldots, W_{31}\}$
2. Used words from the message. $Y_1 = \{M_0, M_1, M_4, M_5, M_8, M_9, M_{12}, M_{13}\}$
3. Unused words from the extension part. $X_2 = \{W_{18}, W_{19}, W_{22}, W_{23}\}$
4. Unused words from the message. $Y_2 = \{M_2, M_3, M_6, M_7, M_{10}, M_{11}, M_{14}, M_{15}\}$

Again if we fix the values of the set X_1 we can search trough the values of the set X_2 in order to find collisions in the set Y_1. Since now the total number of bits of the variables in the set Y_1 is $8 \times 32 = 256$ (resp. $8 \times 64 = 512$) from the birthday paradox we can expect that after $\approx 2^{128}$ (resp. $\approx 2^{256}$) attempts of different values from the set X_2 we will find a collision in the set Y_1.

However, the total number of different choices for the set X_2 is exactly $2^{4\times 32} = 2^{128}$ (resp. $2^{4\times 64} = 2^{256}$) so this attack has the same complexity as the brute-force collision search.

3.5 Finding Collisions in Full Turbo SHA-2

We will discuss the strength of the iterated hash function Turbo SHA-2 as a collision resistant function in the light of the known successful attacks against members of MDx family of hash functions.

Finding collisions in MDx family of hash functions has so far always been based on the following principles: Setting up some system of equations obtained from the definition of the hash function, then tracing forward and backward some initial bit differences that will result in fine tuning and annulling of those differences and finally obtaining collisions. But this strategy, directly applied on SHA-2 is not successful. The reasons why it is not successful on SHA-2 is that message expansion of SHA-2 compared with SHA-1 and other predecessor hash functions is nonlinear and much more complex.

The message expansion of Turbo SHA-2 is also nonlinear, but it has greater complexity than SHA-2. In form of equations the message expansion is represented in (1).

$$\left\{ \begin{array}{ll}
W_{16} = & W_0 + \sigma_0(W_1) + W_2 + \sigma_1(W_3) + W_4 + \sigma_0(W_5) + W_6 + \sigma_1(W_7) + \\
 & + W_8 + W_9 + \sigma_0(W_{10}) + W_{11} + \sigma_1(W_{12}) + W_{13} + \sigma_1(W_{14}) + \sigma_0(W_{15}) + \\
 & + P_0 \\
W_{17} = & W_1 + \sigma_0(W_2) + W_3 + \sigma_1(W_4) + W_5 + \sigma_0(W_6) + W_7 + \sigma_1(W_8) + \\
 & + W_9 + W_{10} + \sigma_0(W_{11}) + W_{12} + \sigma_1(W_{13}) + W_{14} + \sigma_1(W_{15}) + \sigma_0(W_{16}) + \\
 & + P_1 \\
\vdots & \\
W_{31} = & W_{15} + \sigma_0(W_{16}) + W_{17} + \sigma_1(W_{18}) + W_{19} + \sigma_0(W_{20}) + W_{21} + \sigma_1(W_{22}) + \\
 & + W_{23} + W_{24} + \sigma_0(W_{25}) + W_{26} + \sigma_1(W_{27}) + W_{28} + \sigma_1(W_{29}) + \sigma_0(W_{30}) + \\
 & + P_{15}
\end{array} \right. \tag{1}$$

If we compare the SHA-2 message expansion with the Turbo SHA-2 message expansion then we can see that Turbo SHA-2 for every new working variable performs summation of seventeen variables modulo 2^{32} (resp. 2^{64}) instead of summation of four variables in SHA-2. It also performs eight times the functions σ_0 and σ_1 instead of two times in SHA-2. Besides, it is a bijective function on the set $\{0,1\}^{512}$ (resp. $\{0,1\}^{1024}$), and every new working variable depends nonlinearly on all previous 16 variables. Then, the last obtained eight 32-bit (resp. 64-bit) variables obtained in the expansion part are used for the definition of the starting values in the iterative part. That guarantees that the complexity of the Boolean functions that represent each of the hash bits as functions of message bits, to behave like random Boolean functions for all eight iterative rounds.

Moreover, eight rounds in the iterative part produce values that are cyclicly dependable on all thirty-two working variables.

Those are the arguments on which we base our claims that Turbo SHA-2 will resist all known collision attacks, i.e. that the workload for finding collisions is $O(2^{\frac{n}{2}}), n = 256, 512$.

3.6 Finding Preimages and Second Preimages of Turbo SHA-2

From the definition of Turbo SHA-2 (similarly as with SHA-2) it follows that from a given hash digest it is possible to perform backward iterative steps by guessing values that represent some relations between working variables of the extension part. For that purpose let us use the following notation:

- The initialization of the variables $a = H_0^{(i-1)} + W_{31}, \quad b = H_1^{(i-1)} + W_{30}, \quad \ldots, \quad h = H_7^{(i-1)} + W_{24}$ will be denoted as the following system:

$$\begin{cases} a_{-1} = H_0^{(i-1)} + W_{31} \\ \vdots \\ h_{-1} = H_7^{(i-1)} + W_{24} \end{cases} \tag{2}$$

- For every iterative round $t = 0, 1, \ldots, 7$, variables that are on the left side of the assignment (equality sign '=') will be denoted by $a_t, b_t, \ldots, h_t$ while variables that are on the right side of the assignment will be denoted by $a_{t-1}, b_{t-1}, \ldots, h_{t-1}$.

With that notation we can write the backward recurrence expressions as it is done in Table 5:

In Table 5 the variables C_{t+1} satisfy the following system of equations:

$$\begin{cases} C_7 = (W_7 \oplus W_{23}) + (W_{11} \oplus W_{31}) + (W_{15} \oplus W_{27}) + W_{19} \\ C_6 = (W_6 \oplus W_{22}) + (W_{10} \oplus W_{30}) + (W_{14} \oplus W_{26}) + W_{18} \\ C_5 = (W_5 \oplus W_{21}) + (W_9 \oplus W_{29}) + (W_{13} \oplus W_{25}) + W_{17} \\ C_4 = (W_4 \oplus W_{20}) + (W_8 \oplus W_{28}) + (W_{12} \oplus W_{24}) + W_{16} \\ C_3 = (W_3 \oplus W_{19}) + (W_7 \oplus W_{27}) + (W_{11} \oplus W_{23}) + W_{15} \\ C_2 = (W_2 \oplus W_{18}) + (W_6 \oplus W_{26}) + (W_{10} \oplus W_{22}) + W_{14} \\ C_1 = (W_1 \oplus W_{17}) + (W_5 \oplus W_{25}) + (W_9 \oplus W_{21}) + W_{13} \\ C_0 = (W_0 \oplus W_{16}) + (W_4 \oplus W_{24}) + (W_8 \oplus W_{20}) + W_{12} \end{cases} \tag{3}$$

Now, we can treat the three systems of equations (1), (2) and (3) as a one system of 32 equations with 32 unknown variables. It is a highly nonlinear system over GF(2) or over $\mathbb{Z}_{2^{32}}$ (resp. $\mathbb{Z}_{2^{64}}$). We can use the fact that the message expansion part is a bijection over $\{0,1\}^{512}$ (resp. $\{0,1\}^{1024}$) and by guessing eight values for $C_0, \ldots, C_7$ we will get the values for $a_{-1}, \ldots, h_{-1}$, and from (2) we can get the values for $W_{31}, \ldots, W_{24}$. In such a way we can reduce the size of the system to 24 equations (equations (1) and (3)) with 24 unknown variables $W_0, \ldots, W_{23}$. We can further guess eight more

```
1. Initialize eight variables $a_7$, $b_7$, $c_7$, $d_7$, $e_7$, $f_7$, $g_7$ and $h_7$.

2. For t=6 to -1
   {
       $T_2 = \sum_0(b_{t+1}) + Maj(b_{t+1}, c_{t+1}, d_{t+1})$
       $T_1 = a_{t+1} - T_2$
       $a_t = b_{t+1}$
       $b_t = c_{t+1}$
       $c_t = d_{t+1}$
       $d_t = e_{t+1} - T_1$
       $e_t = f_{t+1}$
       $f_t = g_{t+1}$
       $g_t = h_{t+1}$
       $h_t = T_1 - \sum_1(e_t) - Ch(a_t, f_t, g_t) - C_{t+1}$
   }
```

Table 5. Backward recurrence expressions of Turbo SHA-2. Note that the relations for the variables C_{t+1} are given in (3).

variables $(W_{16}, \ldots, W_{23})$ and then solve the system (1) of 16 equations with 16 unknowns, as it is done in the proof of Theorem 1. However, in such a case the obtained solutions $W_0, \ldots, W_{15}$ will additionally have to satisfy the system of equations (3).

In this moment there exists no mathematical theory that will successfully solve differential equations of addition with more than two variables.

Recently Paul and Preneel [24] have successfully solved the problem of finding solutions in polynomial time of differential equations of addition with two variables x and y of type $(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$ where α, β and γ are constants. Someone can use their algorithm to try to solve the system (3). The problem is that their algorithm is for equations with two variables, and their strategy extended to solving systems of differential equations of addition with three or more variables has exponential complexity i.e. is of the order $O(2^{b \times k})$ where b is the bit length of the variables, and k is the number of equations. That means that their strategy for solving the system (3) has a complexity $O(2^{32\times 8}) = O(2^{256})$ (resp. $O(2^{64\times 8}) = O(2^{512})$).

On the other hand, with just a random guess of the variables $W_{16}, \ldots, W_{23}$, we have a probability of 2^{-n}, $(n = 256, 512)$ that the obtained solution $W_0, \ldots, W_{15}$ satisfies also the system of equations (3).

Those are the arguments on which we base our claims that Turbo SHA-2 will resist all known preimage and second preimage attacks, i.e. that the workload for finding preimages and second preimages has the complexity of $O(2^n), n = 256, 512$.

4 Implementation

We have used the SHA-2 optimized C implementation of Dr. Brian Gladman, [12] and just implemented our design principles described in this paper.

A comparison between SHA-2 and Turbo SHA-2 in machine cycles per data byte on AMD and Intel processors for different hash data lengths are given in Table 6. We can see that the speed of Turbo SHA-2 is two to eight times faster than SHA-2 on different processors.

AMD64 (64-bit mode) Bytes processed	SHA-224	Turbo SHA-224	SHA-256	Turbo SHA-256	SHA-384	Turbo SHA-384	SHA-512	Turbo SHA-512
1	1436.0	853.0	1483.0	871.0	1864.0	979.0	1939.0	1056.0
10	145.3	86.8	149.9	88.6	187.9	99.6	195.6	107.2
100	27.9	16.2	28.4	16.4	19.9	11.2	20.6	11.8
1,000	21.1	11.7	21.1	11.7	13.9	6.6	14.0	6.7
10,000	20.4	11.2	20.4	11.2	13.5	6.2	13.5	6.2
100,000	20.4	11.2	20.4	11.2	13.4	6.2	13.4	6.2

AMD64 (32-bit mode) Bytes processed	SHA-224	Turbo SHA-224	SHA-256	Turbo SHA-256	SHA-384	Turbo SHA-384	SHA-512	Turbo SHA-512
1	1608.0	955.0	1628.0	984.0	7246.0	2676.0	7487.0	3896.0
10	161.9	96.6	163.9	99.5	725.4	268.4	749.8	390.4
100	31.3	17.8	31.8	18.1	73.7	27.9	75.8	40.1
1,000	23.8	12.7	24.1	12.7	54.2	17.5	54.4	18.7
10,000	23.2	12.3	23.4	12.3	52.9	16.7	52.9	16.7
100,000	23.6	12.6	23.3	12.6	52.6	16.8	52.5	16.8

Intel P3(32-bit mode) Bytes processed	SHA-224	Turbo SHA-224	SHA-256	Turbo SHA-256	SHA-384	Turbo SHA-384	SHA-512	Turbo SHA-512
1	2865.0	1290.0	2993.0	1316.0	23253.0	3369.0	23653.0	3582.0
10	294.1	129.8	292.5	132.4	2380.1	338.9	2433.7	360.1
100	59.4	24.5	55.8	24.8	241.9	35.5	239.2	37.7
1,000	42.7	17.6	42.7	17.6	177.9	22.2	177.5	22.4
10,000	41.4	17.0	41.5	17.0	174.5	21.1	174.7	21.3
100,000	41.0	16.9	41.0	16.9	173.1	20.7	172.8	20.7

Intel P4(32-bit mode) Bytes processed	SHA-224	Turbo SHA-224	SHA-256	Turbo SHA-256	SHA-384	Turbo SHA-384	SHA-512	Turbo SHA-512
1	3153.0	1997.0	3193.0	2009.0	11461.0	5757.0	11949.0	5981.0
10	315.3	199.7	320.5	200.9	1146.1	573.7	1179.3	598.9
100	63.9	38.0	61.7	38.0	118.0	60.1	121.5	63.8
1,000	48.7	27.8	47.5	28.0	84.6	37.3	84.9	37.6
10,000	45.9	27.0	45.4	27.0	82.3	35.5	82.3	35.5
100,000	44.3	26.8	44.7	27.0	81.3	34.9	81.3	34.9

Intel Core 2 Duo (32-bit mode) Bytes processed	SHA-224	Turbo SHA-224	SHA-256	Turbo SHA-256	SHA-384	Turbo SHA-384	SHA-512	Turbo SHA-512
1	1603.0	973.0	1621.0	991.0	5923.0	2521.0	6157.0	2710.0
10	159.4	97.3	163.0	98.2	594.1	255.7	616.6	279.1
100	30.9	17.9	31.2	18.4	60.7	26.6	62.8	28.4
1,000	23.1	12.6	23.1	12.6	43.3	16.5	43.5	16.7
10,000	22.4	12.2	22.4	12.2	42.1	15.7	42.1	15.7
100,000	22.3	12.1	22.3	12.1	41.7	15.5	41.7	15.5

Table 6. Speed comparison between SHA-2 and Turbo SHA-2

5 Conclusion

In this paper we have constructed a new family of cryptographic hash functions called Turbo SHA-2, based on the hash functions of the family SHA-2. We have introduced two novelties in the design of Turbo SHA-2: 1. Computations in the iterative part of the compression function start by using variables produced in the message expansion part that have the complexity level of a random Boolean function, and 2. Variables produced in the message expansion part are not discarded after the processing of the current message block, but are used for the construction of the three times wider chain for the next message block.

These two novel principles enabled us to build a compression function of Turbo SHA-2 that has a message expansion part with just 16 new variables. The iterative part has just 8 rounds, the function has three times more chaining variables, it is more robust and resistant against generic multi-block collision attacks, it is resistant against generic length extension attacks and it is 2 - 8 times faster than original SHA-2.

References

1. E. Biham and R. Chen, "Near-collisions of SHA-0," Cryptology ePrint Archive, Report 2004/146, 2004. http://eprint.iacr.org/2004/146
2. B. den Boer, and A. Bosselaers: "An attack on the last two rounds of MD4", CRYPTO 1991, LNCS, 576, pp. 194-203, 1992.
3. B. den Boer, and A. Bosselaers: "Collisions for the compression function of MD5", EUROCRYPT 1993, LNCS 765, pp. 293-304, 1994.
4. F. Chabaud and A. Joux, "Differential collisions in SHA-0," Advances in Cryptology, Crypto98, LNCS, vol.1462, pp.56-71, 1998.
5. J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya: "Merkle-Damgård revisisted: How to construct a hash function," CRYPTO 2005, LNCS 3621, 2005.
6. I.B. Damgård, "A design principle for hash functions", CRYPTO 1989, LNCS 435, pp. 416-427, 1990.
7. H. Dobbertin: "Cryptanalysis of MD4", J. Cryptology 11, pp. 253-271, 1998.
8. H. Englund, T. Johansson and M. S. Turan, " A Framework for Chosen IV Statistical Analysis of Stream Ciphers", INDOCRYPT 2007, in press.
9. E. Filiol, "A New Statistical Testing for Symmetric Ciphers and Hash Functions", Proc. ICICS 2002, LNCS 2513, pp. 342-353, 2002.
10. P. Gauravaram, W. Millan and J. G. Nieto: "Some thoughts on Collision Attacks in the Hash Functions MD5, SHA-0 and SHA-1", Cryptology ePrint Archive: Report 2005/391.
11. H. Gilbert and H. Handschuh, "Security analysis of SHA-256 and sisters", Selected Areas in Cryptography'03 (SAC 2003), LNCS 3006, pp. 175–193, 2003.
12. Dr. B. Gladman, "SHA1, SHA2, HMAC and Key Derivation in C", http://fp.gladman.plus.com/cryptography_technology/sha/index.htm
13. P. Hawkes, M. Paddon and G. G. Rose, "On corrective patterns for the SHA-2 family", Cryptology ePrint Archive, Report 2004/207, August 2004, http://eprint.iacr.org/
14. A. Joux: "Multicollisions in iterated hash functions. Application to cascaded constructions", CRYPTO 2004, LNCS 3152, pp 306-316, 2004.
15. J. Kelsey and B. Schneier: "Second preimages on n-bit hash functions for much less than 2^n work," EUROCRYPT 2005, LNCS 3494, pp 474-490, 2005.
16. S. Lucks, "Design Principles for Iterated Hash Functions", Cryptology ePrint Archive, report 2004/ 253.
17. F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen, "Analysis of step-reduced SHA-256", Fast Software Encryption - FSE06, LNCS 4047, pp. 126-143, 2006.
18. K. Matusiewicz, J. Pieprzyk, N. Pramstaller, C. Rechberger, and V. Rijmen, "Analysis of simplified variants of SHA-256", In Proceedings of WEWoRC 2005, Lecture Notes in Informatics (LNI) - Vol. P-74, ISBN 3-88579-403-9, pp. 123 – 140, 2005.
19. R. Merkle, "One way hash functions and DES," CRYPTO 1989, LNCS 435, pp. 428–446, 1990.
20. NIST, Secure Hash Signature Standard (SHS) (FIPS PUB 180-2), United States of American, Federal Information Processing Standard (FIPS) 180-2, 2002 August 1.

21. NIST, First Cryptographic Hash Workshop, October 31 - November 1, 2005, Second Cryptographic Hash Workshop, August 24-25, 2006, http://csrc.nist.gov/groups/ST/hash/first_workshop.html, http://csrc.nist.gov/groups/ST/hash/second_workshop.html.
22. NIST Tentative Timeline for the Development of New Hash Functions, http://csrc.nist.gov/groups/ST/hash/timeline.html
23. S. O'neil, "Algebraic structure defectoscopy", ECRYPT Special Workshop, Tools for Cryptanalysis Krakow (Poland), September 2425, 2007, http://eprint.iacr.org/2007/378
24. S. Paul and B. Preneel, "Solving Systems of Differential Equations of Addition", ACISP 2005, LNCS 3574, pp. 75-88, 2005.
25. B. Preneel: "Analysis and design of Cryptographic Hash Functions," PhD thesis, Katholieke Universiteit Leuven, 1993.
26. B. Preneel, R. Govaerts and J. Varitlevalle: "Boolean functions satisfying higher order propagation criteria", EUROCRYPT 1991, LNCS 547, pp. 141–152, 1991.
27. B. Preneel: "The State of Cryptographic Hash Functions", Lectures on Data Security, LNCS 1561, pp. 158–182, 1999.
28. M.-J. O. Saarinen, "Chosen-IV Statistical Attacks on eStream Ciphers", Proceeding of SECRYPT 2006, pp. 260 – 266, 2006.
29. S. Vaudenay, "On the need for multipermutations: Cryptanalysis of MD4 and SAFER", Fast Software Encryption - FSE95, LNCS 1008, pp. 286–297, 1995.
30. X. Wang, X. Lai, D. Feng, H. Chen and X. Yu, "Cryptanalysis of the Hash Functions MD4 and RIPEMD", EUROCRYPT 2005, LNCS 3494, pp. 1–18, 2005.
31. X. Wang and H. Yu , "How to Break MD5 and Other Hash Functions", EUROCRYPT 2005, LNCS 3494, pp. 19–35, 2005.
32. X. Wang, H. Yu, Y. L. Yin "Efficient Collision Search Attacks on SHA-0", CRYPTO 2005, LNCS 3621, pp. 1–16, 2005.
33. X. Wang, Y. L. Yin, H. Yu, "Collision Search Attacks on SHA-1", CRYPTO 2005, LNCS 3621, pp. 17–36, 2005.
34. H. Yoshida and A. Biryukov, "Analysis of a SHA-256 variant", Selected Areas in Cryptography'05 (SAC 2005), LNCS 3897, pp. 245–260, 2005.
35. G. Yuval, "How to swindle Rabin", Cryptologia, 3, pp. 187-190, 1979.

Appendix 1: Definition of a monomial test NANT (Normalized Average Number of Terms).

Let $n \geq r \geq 1$ be integers and let $F : \{0,1\}^n \to \{0,1\}^r$ be a vector valued Boolean function. The vector valued function F can be represented as an r-tuple of Boolean functions $F = (F^{(1)}, F^{(2)}, \dots, F^{(r)})$, where $F^{(s)} : \{0,1\}^n \to \{0,1\}$ $(s = 1,2,\dots,r)$, and the value of $F^{(s)}(x_1,\dots,x_n)$ equals the value of the s-th component of $F(x_1,\dots,x_n)$. The Boolean functions $F^{(s)}(x_1,\dots,x_n)$ can be expressed in the Algebraic Normal Form (ANF) as polynomials with n variables $x_1,\dots,x_n$ of kind $a_0 \oplus a_1x_1 \oplus \dots \oplus a_nx_n \oplus a_{1,2}x_1x_2 \oplus \dots \oplus a_{n-1,n}x_{n-1}x_n \oplus \dots \oplus a_{1,2,\dots,n}x_1x_2\dots x_n$, where $a_\lambda \in \{0,1\}$. Each ANF have up to 2^n terms (i.e. monomials), depending of the values of the coefficients a_λ. Denote by $L_{F^{(s)}}$ the number of terms in the ANF of the function $F^{(s)}$. Then the number of terms of the vector valued function F is defined to be the number $L_F = \sum_{s=1}^{r} L_{F^{(s)}}$.

Definition 1. *Let $F : \{0,1\}^n \to \{0,1\}^r$ be a vector valued Boolean function. For any $k \in \{1,\dots,n\}$ and any assembly of S subsets $\sigma_j = \{i_1, i_2, \dots, i_k\} \subset \{0,1,\dots,n-1\}$ chosen uniformly at random $(1 \leq j \leq S)$, let F_{σ_j} denote the restriction of F defined by $F_{\sigma_j}(x_1,x_2,\dots,x_n) = F(0,\dots,0,x_{i_1},0,\dots,0,x_{i_2},0,\dots,0,x_{i_k},0,\dots,0)$. We define a random variable $\overline{L_F}$ – the Normalized Average Number of Terms (NANT) as:*

$$\overline{L_F} = \overline{L_F}(r,k) = \frac{1}{r} \cdot \frac{1}{2^{k-1}} \cdot \lim_{S\to\infty} \frac{1}{S} \sum_{j=1}^{S} L_{F_{\sigma_j}}.$$

Since the subsets σ_j are chosen uniformly at random, the average values of $L_{F^{(s)}_{\sigma_j}}$ $(s = 1,2,\dots,r)$ are 2^{k-1} and the average value of $L_{F_{\sigma_j}}$ is $r2^{k-1}$. Also, $L_{F^{(s)}_{\sigma_j}} \leq 2^k$. So, the following theorem is true:

Theorem 2. *For any function $F : \{0,1\}^n \to \{0,1\}^r$ chosen uniformly at random from the set of all such functions, for any value of $r \geq 1$ and for any $k \in \{1,\dots n\}$, it is true that*

$$0 \leq \overline{L_F} \leq 2$$

and that the expected value is

$$EX(\overline{L_F}) = 1.$$

□